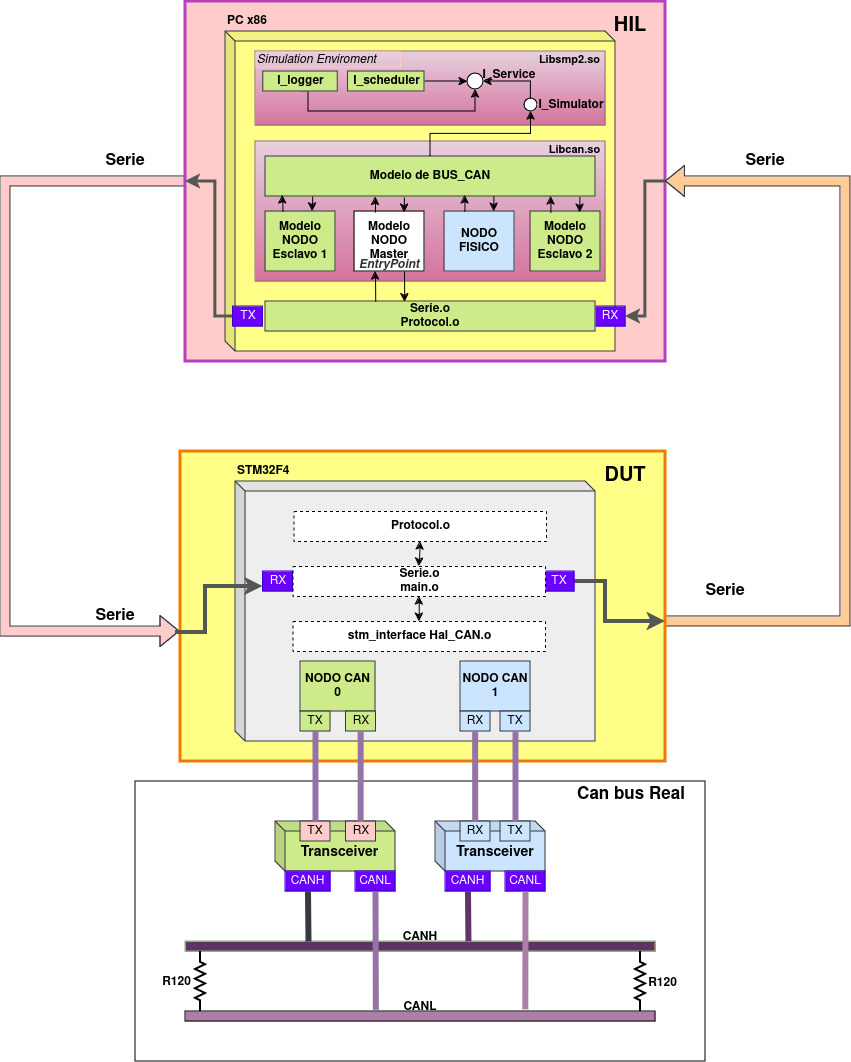
***Descripción del la simulación HIL con SMP2***

En el siguiente diagrama se puede observar el DUT(dispositivo bajo prueba) y el simulador. Donde el simulador esta interactuando con el sistema abajo prueba en un enlace HIL(hardware in the loop).

Lo primero que tenemos que observar es que las entradas y salidas del DUT, estan conectadas al simulador, donde la idea principal es inyectar señales al DUT, y tambien obtener las salidas del sistema.

El funcionamiento del DUT, consiste en crear y controlar una red CAN, esta genera 2 nodos, NODO 0 y NODO 1, con sus respectivos Transceivers. La configuración del NODO 0(nodo Máster), es configurado en forma estática en la creación del firmware, pero la configuración del NODO1(esclavo), es a través de la información del payload de un mensaje CAN virtual llamado NODO\_MASTER, creado bajo SMP2. Es decir el id, y payload del NODO 1, lo obtiene desde el entorno virtual de la red CANBus en smp2. Luego el NODO 1, enviá en su payload la información a la redCANBus virtual, donde el modelo de BUSCAN publica los modelos de NODOS virtuales(NODO\_MASTER, NODO\_E, NODO\_M), y el NODO 1(FÍSICO), donde se puede ver reflejado su id y payload.

Se observa con el mismo color el NODO 1(en el DUT) y del NODO FÍSICO(Pc) en el modelo de red virtual BUSCAN, denotando que es el mismo NODO. La comunicación entre los dos sistemas se realiza con el protocolo SERIE. Esta comunicacion tiene un manejo de los bytes con un protocolo agregado(llamado PROTOCOL), para la organización(empaquetado de variables, id, payload,crc, etc).



Para la compilación, enlazado y gestión de dependencias del proyecto se utilizo la herramienta CMake, donde se genera alguno scripts(CMakelist.txt), que se cargan los diferentes archivos fuentes .cpp(lenguaje c++), que generan los respectivos archivos binario .o(es el lenguaje de maquina, son los archivo objetos compilados), como ademas las librerías.

Los siguientes archivos fuentes generan los respectivos archivos binarios:

Api.cpp →Api.o

Serie.cpp →Serie.o

Protocolo.cpp → Protocol.o

Estos se linkean(enlazan) con los fuentes de Libsmp2(biblioteca compartida .so), que generan la biblioteca compartida Libcan.so, que es un archivo compilado a través del comando cmake .. ubicado en una carpeta llamada Build, utilizando comando make para ejecutar el script.

La librería libsmp2 que usa libcan como dependencia, la cual es la encargada de proveer los servicios de simulación y los modelos de componentes abstractos. Esta librería libsmp2 está desarrollada en C++. Ofrece la posibilidad de reutilizar el código en los diferentes simuladores que se lleven a cabo, con la garantía de que ya fue testeado y corregido en desarrollos previos a libcan. Al extender las clases de la librería libcan(modelos de nodos y red can), heredan la funcionalidad del estándar, de esta forma se cuenta con la implementación de problemas comunes de modelado. Como consecuencia de esta decisión de diseño se obtienen modelos portables que pueden reutilizarse en otras soluciones de simulación que implementen SMP2.

El estándar SMP2 propone una arquitectura modular. La misma cubre dos tipos de componentes:

la Simulación con instancias de Modelos que proveen el comportamiento específico a la aplicación,

y el Ambiente de Simulación que provee los servicios de simulación. Esta arquitectura se encuentra

implementada en la librería libsmp2 de la siguiente manera:

Los servicios que propone la arquitectura implementan la interface Smp::IService:

 **Logger:** Permite el logging de mensajes de diferentes tipo.

 **Scheduler:** El scheduler es el encargado de llamar a los entry points basados en eventos

temporales o cíclicos, depende fuertemente del tiempo marcado por el Time Keeper.

Implementa Smp::Services::IScheduler.

 **TimeKeeper:** Este servicio provee cuatro tipos de tiempos del SMP2. Estos son: tiempo de

simulación relativo, tiempo de época absoluto, tiempo relativo de misión y el tiempo

relacionado al reloj de la computadora. Implementa el servicio Smp::Services::ItimeKeeper.

 **EventManager:** Los componentes pueden registrar entry points a un evento global a través de manejadores de eventos, se puede realizar broadcast de eventos y definir nuevos tipos. Para que una función pueda ser considerada entry point y de esta manera pueda ser agregada en el Scheduler o en el EventManager, debe ser una instancia de IEntryPoint y no debe poseer parámetros ni valor de

retorno.

Una instancia de ISimulator lanza un nuevo thread donde correrá la simulación. ISimulator provee

las interfaces de control de la simulación, como son Run(), Hold(), etc., y además del acceso a los

modelos y servicios mencionados previamente. La simulación avanza de a ciclos de simulación que

el TimeKeeper controla y en cada ciclo se ejecutan una serie de EntryPoints planificados por el

Scheduler. Un EntryPoint es una tarea que se registra en el ciclo de simulación. Al agregarse en el

Scheduler se indica con qué periodicidad éste deberá ejecutarla y en qué orden.

Para generar un modelo para la simulación es necesario que una clase implemente la interface

IManagedModel para dar acceso de sus atributos al manejador de modelos entre los que pueden

encontrarse los EntryPoints.

**Desarrollo del funcionamiento de la arquitectura**

Siguiendo esta arquitectura se creó libcan que hereda de la clase ISimulator y es, para la

aplicación, el ambiente de simulación. El mismo implementa los métodos Configure(), Publish() y

Connect(). Cada uno encargado de hacer broadcast de estos métodos a los modelos que

componen el simulador. Estos métodos son indispensables para la inicialización del ambiente,

obteniendo como resultado final la simulacion.

 Método Publish(): Este método informa al entorno de simulación que se crearon nuevas

instancias de modelos y que pueden ser publicadas. Con este método el entorno de

simulación le indica a cada modelo que se publique. Es llamado por la aplicación luego de

crear todos los modelos.

 Método Configure(): Informa al ambiente de simulación que todas las instancias de modelo

fueron configuradas y que puede ejecutarse su propia configuración. Con este método el

entorno de simulación le indica a cada modelo que se configure. Este método es llamado

por la aplicación luego de publicar todos los modelos.

 Método Connect(): Informa al ambiente de simulación que se creó y configuró la jerarquía

de modelos y que pueden conectarse al simulador. El entorno le indica a cada modelo que

se conecte con el simulador. Se llama desde la aplicación luego de publicar y configurar

todos los modelos.

El simulador que heredan de clases de la libsmp2, poseen acceso al Scheduler. El mismo le asigna un momento de ejecución para cada procedimiento (entry points) de la simulación.

Los entry points actúan como tareas que se ejecutan periódicamente al registrarlas en el

Scheduler. La clase Nodo del archivo api.o posee el entry point, pero cuando el Nodo es Master, y es quien asigna la velocidad real de simulación que se está alcanzando en base a las configuraciones y las capacidades del HW. esta clase, luego hace publicar los nodos en la red can.

Entonces la clase Nodo es la encargada de registrar sus tareas en el Scheduler, por lo que el ciclo de ejecución queda de la siguiente manera:

**DIAGRAMA UML(aca quiero hacer un unl de secuencia, te parece?**

El testing de este proyecto fue haciéndose en etapas, es decir, primero se

probó la placa Stm32 del funcionamiento del puerto CAN, luego la capa de comunicación con la PC mediante puerto Serie. Luego se comenzo con pruebas en el simulador, practicando con varios ejemplos, luego se emepzo a dar forma a un modelo de Nodo, y a suvez comunicar con la placa Nucleo medienate puerto Serie. Despues se comenzo a publicar los modelos de Nodos en un modelo de Red CAN. Los modelos CAN, como la Red CAN, se visualizan en la terminal del visual estudio code. La principal ventaja de la implementación del sistema a través de librerías, fue que esta evolución fue natural. Los tests sirvieron como retroalimentación del diseño y la codificación ya que para lograr una completa batería de pruebas, las responsabilidades debían estar bien asignadas y las clases desacopladas.